INTERNET-DRAFT
CAT Working Group
Expires January 2001
draft-ietf-cat-acc-cntrl-frmw-04.txt

Tatyana Ryutov
Clifford Neuman
USC/Information Sciences Institute
July 11, 2000

Access Control Framework for Distributed Applications

0. Status Of this Document

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

To view the entire list of current Internet-Drafts, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Northern Europe), ftp.nis.garr.it (Southern Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).

1. Abstract

This document describes a unified model to support authorization in a wide range of applications, including metacomputing, remote printing, video conference, and any other application which will require interactions between entities across autonomous security domains.

The document proposes requirements for the support of:

- flexible and expressive mechanism for representing and evaluating security policies

- uniform authorization service interface for facilitating access control decisions for applications and requesting access control information about a particular resource.

This specification defines structures and their uses at a level independent of underlying mechanism and programming language environment. This document is accompanied by a second one describing the details of proposed structures and services along with bindings for C language environments. This document is to be found in draft-ietf-cat-gaa-cbind-04.txt.

2. Introduction

The variety of services available on the Internet continues to increase and new classes of applications such as metacomputing, remote printing, video conference is evolving. These applications will require interactions between entities across autonomous security domains.

The distributed nature of the system, consisting of mutually suspicious security domains, requires a mechanism, which provides fine-grained access control to resources.

For example, access control requirements of a remote printing application may include:

- authorized individual users and organizations
- time availability, e.g. time of the day or day of the week
- restrictions on resources consumed by the clients, e.g. maximum job size, maximum number of pages per job
- required confidentiality/integrity message protection
- accounting for consumed resources

Access control requirements of large-scale multicast application [4], e.g. corporate video conference may include:

- authorized individual users and organizations
- host properties; users on slow hosts or hosts running the wrong OS will be denied communication
- accounting for consumed resources

Some of the security requirements are common across different applications, while others are more individual.

Access control policies can be formulated in many ways. Administrators Of each domain might use domain-specific policy syntax and heterogeneous implementations of the policies.

It is necessary to define a particular framework applicable for a wide range of systems and applications, which will allow to discuss specific requirements for the representation and evaluation of security policies.
An application developer should not have to give anything up by using this framework, therefore the framework should provide support for integration with existing OS or toolkit provided access control mechanisms.
High-speed implementations should be provided for operations that are

supported natively by the underlying system.

The focus of this framework is based on the following two abstractions:

1) uniform mechanism for representation and evaluation of security policies

It should be capable of implementing a number of different security policies, based on diverse authorization models, which can coexist in distributed system. Standardizing the way that applications define their security requirements provides the means for integration of local and distributed security policies and translation of security policies across multiple authorization models.

The mechanism should support the common authorization requirements but provide the means to defining and integration with application or organization specific policies as well. Applications should not need to re-implement the basic authorization functions in an application-specific manner.

2) Generic Authorization and Access control Application Program Interface (GAA-API)

A common API will facilitate authorization decisions for applications. An application invokes the API functions to determine if a requested operation or set of operation are authorized or if additional checks are necessary.

The API will support the needs of most applications, thus not forcing the developers to design their own authorization mechanisms.
The API will allow better integration of multiple mechanisms with application servers. The GSS API can be used by the GAA-API to obtain principal's identity see section 11.

Section 12 gives an extended example how the GAA-API can be used by applications.

3. Glossary

OBJECT (RESOURCE)
entity that has to be protected e.g. hosts, files.

SUBJECT
entity that can initiate requests to an object e.g. individual users, hosts, applications and groups.

PRINCIPAL
identity associated with a subject as a result of some unspecified authentication protocol. It can refer to a person, group, host, and application. Several principals can be associated with the same subject.

SECURITY POLICY
   the set of rules that govern access to objects

ACCESS RIGHT (OPERATION, PERMISSION)
   a particular type of access to a protected object e.g. read, write,
   and execute

RESTRICTION (CONDITION)
   a specific policy allowing an operation to be performed on an
   object

This policy can have two meanings:

    1) descriptive
         An operation is allowed if certain condition is satisfied.
         For instance, a policy may require concurrence of two
         principals to perform some operation. If participation of
         both principals can be proved then this policy is satisfied.

    2) prescriptive
         An operation will be allowed if certain restrictive policy
         is enforced.

         For example, a process will be authorized to run on a host
         if the memory usage limits that a process can occupy in main
         memory satisfies certain constraints. Continuous evaluation of
         restriction is required.

DELEGATION
   is the ability of a principal to give to another principal limited
   authority to act on its behalf.

CREDENTIAL
   a statement of identity, group membership and non-membership,
   privilege attribute and transfer of privilege encoded in
   certificates.

4. Architecture

The major components of the architecture are:

Authentication mechanisms (perhaps involving an authentication
server) perform authentication of users and supply them with
initial credentials.

A group server is trusted to maintain and provide group membership
information. A group is a convenient method to associate a name with
a set of principals for access control purposes. A group server issues
group membership and non-membership certificates. When a connection is
established with an application server, these certificates are evaluated
(evaluation may be deferred until needed) the results are placed into
the GAA-API security context. They are checked by the GAA-API when

making authorization decisions.

The application calls the GAA-API routines to check authorization against the application authorization model.
These routines obtain access control information from local files, distributed authorization servers, and from credentials provided by the user, combining local and distributed authorizations under a single API according to the requirements of the application.

Delegation is supported through inclusion of delegated credentials. Mechanism for delegation such as those supported by restricted proxies [1] in the security context, where they are available for use by authentication and authorization mechanisms used for subsequent connections from the server (now acting as an intermediary) to another server.

5. Objects

The purpose of access control is protecting objects from unauthorized access. The kinds of objects to be protected are specific to the application to which the authorization model is applied and are not included into the authorization model.
The objects that need to be protected include files, directories, network connections, hosts, auxiliary devices, e.g. printers and faxes, and other entities. An authorization mechanism should support these different kinds of objects in a uniform manner. Same security attribute structure should be used to specify access policies for different kinds of objects.
Object names should be drawn from the application-specific name space and must be opaque to the authorization mechanism.

6. Security policy representation

One may think about the security policy associated with a protected resource as a set of operations which a defined set of principals is allowed to perform on the target resource, and optional constraints placed on the granted operations. For example, a system administrator can define the following security policy to govern access to a printer: "Joe Smith and members of Department1 are allowed to print documents Monday through Friday, from 9:00AM to 6:00PM".
This policy can be described by an ACL mechanism, where for each resource, a list of valid entities is granted a set of access rights.
The same policy can be implemented using a capability mechanism.
However, traditional ACL and capability abstractions should be extended to allow conditional restrictions on access rights.

Therefore, in implementing a policy, it should be possible to define:
1) access identity
2) grantor identity
3) a set of access rights

4) a set of conditions

Policy is represented by a sequence of tokens. Each token consists of:

Token Type

Defines the type of the attribute. Tokens of the same type have the same authorization semantics.

Defining Authority

It indicates the authority responsible for defining the value within the attribute type.

Value

The value of the security attribute. Its syntax is determined by the attribute type. The name space for the value is defined by the Defining Authority field

6.1. Access identity

The access identity represents an identity to be used for access control purposes. The authorization framework should support the following kinds of access identity:

1) USER
identifies a person, e.g. authenticated user name.

2) HOST
identifies a subject as a machine from which request to access the object was originated, e.g., an IP address or host DNS name or host public key.

3) APPLICATION
identifies a certain program that has its own associated identity (principal),e.g., a checksum or certified name.
This can be useful to grant access to a certain application, e.g. payment program, that is trusted to be written correctly and perform only its intended purpose.

4) CA
identifies a Certification Authority responsible for issuing a certificate.

5) GROUP
identifies a group of subjects. The kind of subjects (individual user, host or application) composing the group is opaque to the authorization mechanism.

6) ANYBODY
      represents any subject regardless of authentication.
      This may be useful for setting the default policies.

The framework should support multiple existing principal naming methods. Different administrative domains might use different authentication mechanisms, each having a particular syntax for specification of principals. Therefore, Defining Authority for access identity indicates underlying authentication mechanism used to provide the principal identity. Value represents particular principal identity.

Examples of access identities are:

```
Token Type:         access_id_ANYBODY
Defining Authority: none
Value:              none

Token Type:         access_id_HOST
Defining Authority: IPaddress
Value:              164.67.21.82

Token Type:         access_id_GROUP
Defining Authority: DCE
Value:              15

Token Type:         access_id_CA
Defining Authority: X.509
Value:              /C=US/O=Globus/CN=Globus CA

Token Type:         access_id_APPLICATION
Defining Authority: checksum
Value:              123x56
```

6.2. Grantor identity

The grantor identity represents an identity used to specify the grantor of a capability or a delegated credential. Its structure is similar to the one of the access identity described in the previous subsection.

Example:

```
Token Type:         grantor_id_USER
Defining Authority: kerberos.V5
Value:              kot@ISI.EDU
```

6.3. Access rights

It must be possible to specify which principals or groups of principals are authorized for specific operations, as well as who is explicitly denied authorizations, therefore we define positive and negative access

rights.

All operations defined on the object are grouped by type of access to
the object they represent, and named using a tag.
For example, for a file the following operations are defined:

```
Token Type:         pos_access_rights
Defining Authority: local_manager
Value:              FILE:read,write,execute
```

However, in a bank application, an object might also be a customer
account, and the following set of operation might be defined:

```
Token Type:         pos_access_rights
Defining Authority: local_manager
Value:              ACCOUNT:deposit,withdraw,transfer
```

Internally a tagged bit vector represents access rights.
Each bit in the vector corresponds to an access right.
The tag indicates how the bits in the bit vector are to be interpreted,
in the example above, for the set of rights associated with the tag FILE the
first bit should be interpreted as read, while for the set
associated with tag ACCOUNT, the same bit should be interpreted as
deposit.

6.4. Conditions

The conditions specify the type-specific policies under which an
operation can be performed on an object.

Conditions can be categorized as generic or specific. A condition
is generic if it is evaluated by the access control model. Specific
conditions are application-dependent and usually evaluated by the
application.
Conditions are evaluated recursively. Recursive evaluation of
conditions is done not through recursive GAA-API calls, but though a
recursive evaluation within the GAA-API.

6.4.1. Generic conditions

The following list of generic conditions should not be considered
exhaustive.

a) time

   Time periods for which access is granted, e.g. time of day or
   day of the week

b) location

   Location of the principal. Authorization is granted to the
   principals residing in specific hosts or domains.

c) connection

  c.1. security of the connection

      1) required confidentiality message protection
         This condition specifies a mechanism, or a set of
         mechanisms to be used in confidentiality message protection.

      2) required integrity message protection
         This condition specifies a mechanism, or a set of
         mechanisms to be used in integrity message protection.

  c.2. bandwidth

  c.3. particular network (SAN vs. LAN for a cluster)

  c.4. dialup

d) privilege constraints

  In general, a principal may belong to more than one group.
  By default, principal operates with the union of privileges of
  all groups to which it belongs, as well as all of his individual
  privileges.
  In assigning privileges, one can choose to:

    1) have the subject operate with the privilege of only one group
       at a time. This can be used to reduce privileges as a
       protection against accidents.
       E.g. a person is a member of two groups: PROGRAMMERS and
       SYSTEM_MANAGERS. The person may act with the privileges of the
       group PROGRAMMERS most of the time, and enable privileges of
       the SYSTEM_MANAGERS group only on occasion.

    2) have the subject operate with privileges of several specified
       groups at a time. This condition class allows one to express
      "group A and group B"

    3) endorsement
       Concurrence of several principals to perform some operation.

e) multi-level security constraints

f) payment

  Specifies currency and amount that must be paid prior
  access to an object will be granted.

g) quota

Specifies a currency and a limit. It limits the quantity of a resource that can be consumed or obtained.

h) strength of authentication

Specifies the authentication mechanism or a set of suitable mechanisms, used to authenticate a user.

i) attributes of subjects

This class of conditions defines a set of attributes that must be possessed by subjects in order to get access to the object, e.g. security label.

j) trust constrains

This class of conditions specify constraints on legitimacy of the received certificate chain and the authenticity of the specified keys:

1) Constraints placed in the certificates by CA when issuing a certificate. They are represented as a list of certificate extensions and marked as critical/non-critical. For example, the extension may contain policy specifying that no further delegation of CA authority is allowed.

2) Specify trustworthiness of CA to sign certificates for limited set of users.

3) Restrictions on certification path:

a) Permitted and denied subtrees of CA authorities.

b) Limit the length of certificate chains (depth of the subtree).

4) Restrictions imposed on cryptographic attributes:
- accepted signature scheme (e.g. DSA/SHA, RSA/MD2)
- minimum public key length (768, 1024 bits)

5) Deciding to trust particular end-entity certificate for particular purpose (operation).

k) audit information

Auditing access to protected objects such as files, print queue that is handing sensitive
documents or a terminal to catch attempted password grabbers, has to be selective since
access requests can occur frequently.

Audit condition enables generating audit records in response to access requests to protected

objects. This condition may specify:

a) When to log:
   - success or failure (or both) of the request
   - monitor particular user's behavior
   - logging in from a number of terminals
   - logging in at unusual times of the day or the week

b) Where to log: write audit information to operator terminal or audit log

6.4.2. Application-specific conditions

If generic conditions are not sufficient for expressing application-specific security policies, applications specify their own conditions.
Anything that can be expressed as type : value alphanumeric string can be a condition. The application must provide a means for evaluation of the application-specific conditions.

```
Token Type:         printer_load
Defining Authority: PrinterManager
Value:              10
```

7. Capabilities and ACLs

Proposed security attributes allow to implement both capabilities and ACLs. For example, the following sequence of security attributes implements an ACL, stating that anyone authenticated by Kerberos.V5 has read access to the targeted object and any member of group 15 connecting from the USC.EDU domain has read and write access to the object.

```
Token Type:         access_id_ANYBODY
Defining Authority: none
Value: none

Token Type:         pos_access_rights
Defining Authority: local_manager
Value:              FILE:read

Token Type:         authentication_mechanism
Defining Authority: system_manager
Value:              kerberos.V5

Token Type:         access_id_GROUP
Defining Authority: DCE
Value:              15

Token Type:         pos_access_rights
Defining Authority: local_manager
Value:              FILE:read,write
```

```
Token Type:         location
Defining Authority: system_manager
Value:              *.USC.EDU
```

The following sequence of security attributes implements
a capability, stating that the capability granted by the group "admin"
grants read access if the capability is presented during the specified
time period.

```
Token Type:         grantor_id_GROUP
Defining Authority: kerberos.V5
Value:              admin@USC.EDU

Token Type:         pos_access_rights
Defining Authority: local_manager
Value:              FILE:read

Token Type:         time_window
Defining Authority: eastern_timezone
Value:              8:00AM-5:00PM
```

8. Ordering Issues

The order in which security attributes are appearing is very important
for correctly interpreting the intended security policy.
It is not clear if interpretation of security attribute ordering should
be included in the draft or left as an implementation issue.

9. Inheritance

Underlying systems may use inheritance, for example the Netware security
model propagates inheritance rights down the directory tree, so that there
is not a single ACL to evaluate. In NT one can get access to an file either
through the directory or through the file itself, so again there are two
ACLs.
Inheritance can be supported in the function one provides to obtain
the ACL (see section 11), i.e. it needs to look at the object name and
determine which sources (possibly multiple sources) to draw ACL entries
from.
It might also add entries that refer to other named ACLs that would be
"included" in the primary ACL or interpreted at other points. How
this should work is something that is open for definition/discussion
with those needing this feature.

10. Security context

The security context is a GAA-API data structure, which is passed as
an argument to the GAA-API.
It stores information relevant to access control policy, e.g.
authentication and authorization credentials presented or used by the
peer entity (usually the client of the request), connection state

information.

The context consists of:

1) Identity

Verified authentication information, such as principal name for a
particular security mechanism.

2) Authorized credentials
   This type of credentials is used to hold delegated credentials and
   capabilities.

3) Group membership
   This type of credentials specifies that the grantee is a member of
   only the listed groups.

4) Group non-membership
   This type of credentials specifies that the grantee is NOT a member
   of the listed groups.

5) Attributes
   This type of credentials contains miscellaneous attributes
   attached to the grantee, e.g. age of the grantee, grantee's security
   clearance.

6) Unevaluated Credentials
   Evaluation of the acquired credentials can be deferred till the
   credential is needed to perform the operation.

7) Evaluation and Retrieval Functions for Upcalls
  These functions are called to evaluate application-specific conditions,
  to request additional credentials and verify them.
  The GSS API is an example of how this can be filled in.

8) Connection State Information
  Contains a mechanism-specific representation of per-connection
  context, some of the data stored here include keyblocks, addresses.

11. Generic Authorization and Access API

The GAA-API is built into applications through a library.
It is used by applications to decide whether a subject
is authorized to perform particular operations on an object. In
this section we provide a brief description of the main GAA-API
routines.

11.1. GAA-API routines

 1) gaa_initialize

   The gaa_initialize must be called before any other GAA-API function.

It initializes the GAA-API structures and defines behavior of the gaa evaluation
routines.

Input:

o Pointer to the GAA-API structure

o Pointer to the implementation-specific structure

The implementation may require implementation-specific information to be passed and
returned during the GAA-API initialization. The structure of the passed and returned
information varies for different underlying implementations, which makes it difficult
to standardize the structure.

Output:

o Status code

2) gaa_cleanup

The gaa_cleanup cleans up internal GAA-API structures allocated and initialized
using the gaa_initialize function. The calling application should call gaa_cleanup
to free memory and internal implementation state before exiting.

Input:

o Pointer to the GAA-API structure

o Pointer to the implementation-specific structure

Output:

o Status code

3) gaa_get_object_policy_info

The gaa_get_object_policy_info function is called to obtain
security policy information associated with the object. In the
ACL-based systems, this information represents object ACLs, in
the capability-based systems, this information may contain a
list of authorities allowed to grant capabilities.

Input:

o Reference to the object to be accessed.

The identifier for the object is from an application-dependent name space, it can be represented as unique object identifier, or symbolic name local to the application.

- Reference to an application-specific authorization database.

Output:

- Status code

- A handle to a structure, containing the security policy associated with the targeted object.

4) gaa_check_authorization

The gaa_check_authorization function tells the application server whether the requested operation or a set of operations is authorized, or if additional checks are required.

Input:

- Pointer to the GAA-API structure.

- A handle to a policy structure, returned by the gaa_get_object_policy_info.

- Principal's security context.

- A list of access rights for authorization.

- GAA-API options structure (optional)
  This argument contains parameters for parameterized operation (see section 13).

Output:

- Status code:

YES code (indicating authorization) is returned if all requested operations are authorized.

NO code (indicating denial of authorization) is returned if at least one operation is not authorized.

MAYBE code (indicating a need for additional checks) is returned if there are some unevaluated conditions and additional application-specific checks are needed, or continuous evaluation is required.

- Detailed answer.

Detailed answer contains:

- Authorization valid time period.
  The time period during which the authorization is granted is returned as condition to be checked by the application.
  Expiration time is calculated based on time-related restrictions expressed by the security attributes and restrictions in the authentication, authorization and delegated credentials.

- The requested operations are returned marked as granted or denied along with a list of corresponding conditions, if any.
  Each condition is marked as evaluated or not evaluated, if evaluated marked as met, not met or further evaluation or enforcement is required. This tells application which policies must be enforced.

- Information about additional security attributes required.
  Additional credentials might be required from clients to perform certain operations, e.g. group membership or delegated credentials.
  The application must understand the conditions that are returned unevaluated or it must reject the request.
  If understood, the application checks the conditions against information about the request, the target object, or environmental conditions to determine whether the conditions are met. Enforcement of the returned conditions is up to the application.

5) gaa_inquire_policy_info

The gaa_inquire_object_policy_info function allows application to discover a particular user's rights on an object. It returns a list of rights that the principal is authorized for and corresponding conditions, if any.

Input:

- Pointer to the GAA-API structure.

- A handle to a policy structure, returned by the gaa_get_object_policy_info.

- Principal's security context.

Output:

- Status code.

- A list of authorized and denied rights and corresponding conditions, if any.

12. Creation of the GAA-API security context

Prior to calling the gaa_check_authorization function, the application must obtain the authenticated principal's identity and store it in the security
context. This context may be constructed from credentials obtained from different mechanisms, e.g. GSS API, Kerberos or others.
This scenario places a heavy burden on the application programmer to provide the integration of the security mechanism with the application. A second scenario is to obtain the authentication credentials from a transport protocol that already has the security context integrated with it. For example, the application can call SSL or authenticated RPC. In this case, it is the implementation of the transport mechanism (usually written by someone other than the application programmer) which calls the security API requesting principal's identity, and which constructs the security context.

The principal's authentication information is placed into the security context and is passed to the GAA-API. When additional security attributes are required for the requested operation, the list of required attributes is returned to be obtained by the application. The application may provide GAA-API with an upcall function for requesting required additional credentials.

The credentials pulled by the GAA-API are verified and added to the security context by the upcall function. A reference to the upcall function is passed to the GAA-API as part of the security context, and it added to the security context by the application or transport.

13. Parameterized operation

Some operations for authorization may require parameters. For example, request to transfer $20 will be expressed as operation: transfer parameter: amount:$20.
Parameters are different from conditions and one can not merge lists of each. Parameters must be typed to describe what they mean, and that condition evaluation functions that check parameters must define the types of the parameters that they check: if it of a single type and what type it is, or if it is a structure, and if it is a structure, what each of the elements in the structure is.
When evaluating conditions that need to see parameters they would then use the GAA options structure and interpret the parameter and match each "variable" with the particular parameter they are looking for.
Data in the GAA options structure is used to pass parameters.

14. Credential evaluation

Credentials are translated to the GAA-API internal format and placed into the GAA-API security context.
When evaluating a policy, e.g., ACL, the necessary credentials are looked for in the security context.

ACL entries containing principals that do not match the current subject identity but grant the requested operation and the identities that are associated with the subject, stored in the security context, e.g. group memberships and delegated credentials, are taken into account when evaluating an ACL.

In general, when an ACL grants requested operation and no additional credentials are required, the GAA-API will look for credentials that can cause denial. A principal may chose to withhold credentials that it believes may result in a denial.
There may be interactions when independent credentials are used, i.e., one set of credentials causes denial, but the other causes accept. Administrator has to deal with these issues by carefully setting policies in an ACL. It may be appropriate to specify more restricted set of rights and require grant credentials to be presented.
A condition may specify whether grant or denial credentials take precedence.

15. Extended example: simple Printer Manager application

To illustrate how the GAA-API is used by application servers we describe a simple Printer Manager application, where protected objects are printers. The Printer Manager accepts requests from users to access printers and invokes the GAA-API routines to make authorization decisions, under the assumption that the administrator of the resources has specified the local policy regarding the use of the resources by means of ACL files.
These files are stored in an Authorization Database, maintained by the Printer Manager.

15.1 Conditions

Administrators will be more willing to grant access to the printers if they can restrict the access to the resources to users or organizations they trust. Further, the administrators should be able to specify time availability, restrictions on resources consumed by the clients and accounting for the consumed resources. To specify these limits, the Printer Manager uses generic conditions, such as time, location, payment and quota.

An example of Printer Manager-specific condition: printer load, expressed as maximum number of jobs that allowed to be submitted to a printer.

15.2 Authorization walk-through

Here we present an authorization scenario to demonstrate the use of the authorization framework for the case of printing a document.
Assume Kerberos V5 is used for principal authentication. Assume that printer A has an ACL stored in the Printer Manager authorization database.

Let's consider a request from user Tom who is connecting from the

ORG.EDU domain to print a document on the printer A at 7:30 PM.

When a client process running on behalf of the user "Tom" contacts the
Printer Manager with the request to submit_print_job (which
is indicated by setting the appropriate bit in the bit vector) to
printer A, the Printer Manager first calls gaa_initialize function
to allocate GAA-API internal structures and initialize the specific
GAA-API state. Then the Printer Manager calls the gaa_get_object_policy_info
function to obtain a sequence of security attributes representing the
security policy expressed in the ACL for the printer A. The upcall
function for retrieving the sequence is passed to the GAA-API and is
being called by the gaa_get_object_policy_info, which returns the
sequence.
Assume that the following sequence was returned:

-- first EACL entry

Token Type: access_id_USER
Defining Authority: kerberos.V5
Value: tom@ORG.EDU

Token Type: pos_access_rights
Defining Authority: PrinterManager
Value: PRINTER:submit_print_job

Token Type: time_window
Defining Authority: pacific_time_zone
Value: 8:00AM-8:00PM

Token Type: printer_load
Defining Authority: PrinterManager
Value: 20

-- second EACL entry

Token Type: pos_access_id_ANYBODY
Defining Authority: none
Value: none

Token Type: pos_access rights
Defining Authority: PrinterManager
Value: PRINTER:view_printer_capabilities

The Printer Manager must place the principal's authenticated identity in
the security context to pass it to the gaa_check_authorization function.
This context may be constructed according to the first or second
scenario, described in section 12.
If Tom is authenticated successfully, then verified identity credentials
are placed into the security context, specifying Tom as the Kerberos
principal tom@ORG.EDU.

Then the Printer Manager calls the gaa_check_authorization function.

In evaluating the security attribute sequence. The set, corresponding to the first entry applies. It grants the requested operation, but there two conditions that must be evaluated.

The first condition time_window : 8AM-8PM is generic and is evaluated directly by the GAA-API. Since, the request was issued at 7:30 PM this condition is satisfied.
The second condition printer_load : 20 is specific.
If the security context defined a condition evaluation function for upcall, then this function is invoked and if this condition is met then the final answer is YES (authorized) and the detailed answer contains: Authorization expiration time : 8PM (assume that authentication credential has expiration time 9PM).

```
Allowed operations:   submit_print_job
List of conditions:   time_window  : 8AM-8PM
                      printer_load : 20
```

Both conditions are marked as evaluated and met.

During the execution of the task the Printer Manager is enforcing the limits imposed on the local resources and authorization time.

If the corresponding upcall function was not passed to the GAA API, the answer is MAYBE and the second condition is marked as not evaluated and must be checked by the Printer Manager.

When additional credential is needed, then if the security context defines a credential retrieval function for upcall, then this function is invoked. If the requested credential is obtained, then the final answer is YES. If the upcall function was not passed to the GAA-API, the answer is NO. Before the Printer Manager exits, it calls gaa_cleanup to free allocated memory and other internal GAA-API state.

16. Integration with existing authentication mechanisms

The framework supports various strengths of user authentication mechanisms. An ACL may have entries associated with different principals identifying the same subject using different authentication methods.
A subject may be granted a different set of rights, depending on the strength of the authentication method used for identification. Specification of weaker authentication methods including network address or username will allow the GAA-API to be used with any existing application that does not have support for strong authentication.

17. Integration with existing authorization mechanisms

Existing tools and ACLs stored by the underlying system should be usable in some sense - perhaps through a converter.

18. Integration with alternative authorization models

The proposed framework can be easily integrated with existing
access control models in a uniform and consistent manner.
[7] shows how this mechanism can implement role-based access control,
Clark-Wilson model and lattice-based policies.

19. References

[1] B.C. Neuman.
Proxy-based authorization and accounting for distributed systems.
Proceedings of the 13th International Conference on Distributed
Computing Systems, Pittsburgh, May 1993.

[2] B.C. Neuman and Theodore Ts'o.
Kerberos: An authentication service for computer networks.
IEEE Communications Magazine, pages 33-38, September 1994

[3] M. Blaze, J. Feigenbaum and J. Lacy.
Decentralized Trust Management.
in Proc. IEEE Symp. on Security and Privacy, IEEE Computer Press,
Los Angeles, pages 164-173, 1996.

[4] Large Scale Multicast Applications (lsma) working group.
Taxonomy of Communication Requirements for Large-scale Multicast
Applications.
Internet draft.

[5] Department of Defense National Computer Security Center.
Department of Defense Trusted Computer system Evaluation Criteria,
December 1985. DoD 5200.28-STD

[6] Tom Parker and Denis Pinkas.
Extended Generic Security Service APIs: XGSS-APIs Access control and
delegation extensions
Internet-Draft IETF Common Authentication Technology WG

[7] T.V. Ryutov and B.C. Neuman
An Authorization Framework for Distributed Systems
Paper was submitted for the NDSS'2000.
http://gost.isi.edu/info/gaa_api.html

20. Acknowledgments

Carl Kesselman, Mike Swift, Denis Pinkas, Gene Tsudik, Brian Tung,
Bapa Rao, Ilia Ovsiannikov and the Xerox IS team have contributed
to discussion of ideas and material in this draft.

21. Authors' Addresses

Tatyana Ryutov
Clifford Neuman

USC/Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90292-6695
Phone: +1 310 822 1511
E-Mail: {tryutov, bcn}@isi.edu